AWS DeepRacer Control

Aakash Parikh, Federik Warburg and Jun Zeng

***Abstract*— The AWS DeepRacer was developed for the research of deep reinforcement learning, however the performance is very limited with training results from the built-in models and variants. To improve the performance of the AWS DeepRacer for tracking trajectories with predefined markers, we explore the viability of utilizing other controllers. We present a simple proportional controller in order to explore the features and functionality of the vehicle. After discussing the dynamical model of the vehicle, a MPC controller based on a reduced bicycle model is also presented. For autonomous driving with this vehicle, we present a standard state estimation and localization of the vehicle based on predefined markers in a given map. Tracking performance with the proportional controller is verified with experiments and the MPC controller is numerically simulated with an expectation of further development. To the best of our knowledge, we are the first ones to develop and open-source a proportional controller and simulated MPC controller for the AWS DeepRacer. A summary of our work and related code can also be found on our website [1] and the GitHub repository [2], respectively.**

I. Introduction

A. Motivation

Deep reinforcement learning has recently been used to control physical systems, instead of model-based techniques like model-predictive control (MPC). As part of this trend, Amazon has recently released their Amazon Web Services (AWS) DeepRacer, a 1/18 scale race car meant for reinforcement learning. The vehicle has a camera, IMU and an onboard computer. In this project, we will implement MPC-based control on the AWS Deepracer and compare its performance to that of Amazon's reinforcement learning based control. The motivation of this project is to understand the strengths and limitations of the two approaches for controlling the vehicle.

B. Related Work

Recent trends in the automotive industry point in the direction of an increase in content of electronics, computers, and controls with an emphasis on the improved functionality and overall system robustness. Autonomous vehicles has grown dramatically in the past decades and recent work [15], [16] has accelerated this research area.

The main components of a modern autonomous vehicle are localization, perception, and control. There are some exiting work [14], [10], [7] about vehicle control focusing on lateral or longitudinal control in the last few decades. In the previous decade, people tended to focus on the control of the vehicles acceleration, brake, and steering using Model Predictive Control (MPC), due to the advances in computing and sensing technologies. Many contributions [2], [4], [9], [8] have been made based on full or simplified, kinematic or dynamical models. Recently, a reinforcement learning approach is introduced in [3] for vehicle cruise control. Moreover, there is interesting research [13], [12] about the intersection of a model-based controller and a learning-based model.

Fig. 1: Amazon DeepRacer

In this year, Amazon developed a a 1/18 scale race car illustrated in Figure 1, which is called as AWS (Amazon Web Services) Deepracer [3] and could be used for the research about reinforcement learning. AWS Deepracer is designed to train its models on Amazon SageWorks [4] and there is a managed machine learning service to integrate the model with the race car. Before the release of this product, there is some practical work about small racing car competition. For example, an algorithm about lane detection was applied in [5], which provides a fast solution with OpenCV that can be used to map a track and ensure that the vehicle is inside the correct path.

C. Contributions

The main contributions of this paper with respect to prior work are:

- Access of camera images and IMU data from ROS topics and pulse-width modulation (PWM) signals.
- Implement a proportional controller on the angle of deviation from the centerline.

A. Parikh and F. Warburg are with the Dept. of Electrical Engineering and Computer Science at UC Berkeley, {aparikh98, warburg}berkeley.edu.

J. Zeng is with the Dept. of Mechanical Engineering at UC Berkeley, {zengjunsjtu}berkeley.edu.

These three authors contribute equally to the work.

[1] https://github.com/Junzeng-x14/AWS-DeepRacer/wiki

[2] https://github.com/Junzeng-x14/AWS-DeepRacer/

[3] https://aws.amazon.com/deepracer/

[4] https://aws.amazon.com/sagemaker/

- Simulate MPC controller for the vehicle's system, which is equivalent to a reduced bicycle model.

The paper is organized as follows. Section II introduces the vehicle's dynamical system - the reduced bicycle model. Section III develops the proportional controller on angle of deviation from centerline and the MPC controller in the inertia frame. Section IV presents the state estimation and localization for the vehicle. The experimental and simulation results are discussed in Section V and VI. Some discussions about advantages and failures of our implementation are presented in Section VII. Finally, Section VIII provides concluding remarks.

II. System Dynamics

To our knowledge, we could only control the system dynamics through the inputs: the vehicle's speed v and the steering angle δ. A reduced bicycle model in the inertial frame can be described as,

$$\dot{x} = v\cos(\phi), \tag{1}$$

$$\dot{y} = v\sin(\phi), \tag{2}$$

$$\dot{\phi} = \frac{v}{l}\tan(\delta), \tag{3}$$

where x and y are coordinates of the center of mass in the inertia frame (X, Y). ϕ is the inertial heading and l represents the length of vehicle from the front axles to the rear axles. Compared to higher fidelity vehicle models, the system identification on the kinematic bicycle model is easier because there are only one parameter to identify: l. This also makes the tuning of controllers easier.

III. Control

We tested and implemented several controllers to make the vehicle to track a given trajectory and we will illustrate them respectively.

A. Proportional Controller

First, we illustrate a simple proportional controller, which has been used to assess and test the performance of the vehicle. Here we define a proportional controller to calculate the steering input and we hold the vehicle's speed to be constant in the forward direction.

$$\delta = K_p e_\alpha, \tag{4}$$

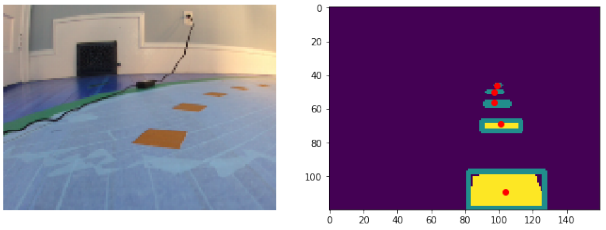
$$v = v_0, \tag{5}$$

where α represents the local angle of deviation from the center line. Given camera output at each step, the raw image is filtered, so that only the color of the center line markers are included in the filtered image, presented in Figure 2a. Bounding boxes are drawn and the centroids of these markers can be found, presented in Figure 2b. Based on this, e_α can be defined as a piece-wise function of angle of deviation, where (x_c, y_c) represents the lowest centroid position and (x_r, y_r) is the bottom middle of the image.

$$e_\alpha(t) = \begin{cases} -1 & \alpha(t) < -0.2, \\ 1 & \alpha(t) > 0.2, \\ 0 & o.w \end{cases} \tag{6}$$

where α can be expressed as,

$$\alpha = \tan^{-1}\left(\frac{y_c - y_r}{x_c - x_r}\right) \tag{7}$$



(a) Raw Camera Input (b) Processed Marker Locations

B. Model Predictive Control

This section presents the motivation of an MPC controller using a reduced bicycle model $\dot{z} = Az + Bu$ described in (1), (2) and (3), where $z = [x; y; \phi]$ and $u = [v; \delta]$ represent the states and the inputs of this system, respectively. A and B can be calculated easily from the linearization,

$$A = 0_{3\times 3} \tag{8}$$

$$B = \begin{bmatrix} \cos(\phi) & 0 \\ \sin(\phi) & 0 \\ \frac{\tan\delta}{l} & \frac{v}{l\cos(\delta)^2} \end{bmatrix} \tag{9}$$

Therefore the forward dynamics can be described as $z_{i+1} = f(z_i, u_i, dt)$ between time steps i and $i+1$,

$$z_{i+1} = A' z_i + B' u_i + C', \tag{10}$$

where $A' = I_{3\times 3} + dtA$, $B' = B'dt$ and $C' = [0, 0, \frac{-v}{l\cos(\delta)^2}]^T dt$. dt represents the C' is added into the forward dynamics to compensate for the error dynamics from the linearization.

At each sampling time t_s, the MPC controller solves the following constrained finite-time optimal control problem:

$$\begin{aligned} \min_U \quad & \sum_{i=0}^{N} (z_i - z_{ref,i})^T Q (z_i - z_{ref,i}) + \sum_{i=0}^{N-1} u_i^T R u_i \\ \text{s.t.} \quad & z_0 = z(t), z_{i+1} = f(z_i, u_i, t_d) \\ & u_{min,i} \le u_i \le u_{max,i} \\ & \dot{u}_{min,i} \le \frac{u_{i+1} - u_i}{t_d} \le \dot{u}_{max,i} \end{aligned} \tag{11}$$

where t_d represents the discretization time for the forward kinematics in the MPC controller. $Q \in \mathbb{R}^3_+$, $R \in \mathbb{R}^2$ and N represents the horizon number. We distinguish the sampling time t_s and the discretization time t_d for the dynamical model of vehicle. $f(z_i, u_i, t_d)$ represent the dynamics update with the reduced kinematic bicycle model through forward Euler, presented in (10).

We don't have too much knowledge about the bounds of vehicle's speed, steering angle (u_{min} and u_{max}) and their derivatives ($\dot{u}_{min}$ and $\dot{u}_{max}$) respectively. For simplification, we pick values from other small race cars [5] as an estimation.

[5] http://www.barc-project.com/

IV. State Estimation

In this section we explain how we were able to localize the absolute position of the vehicle using only monocular camera input and predefined markers.

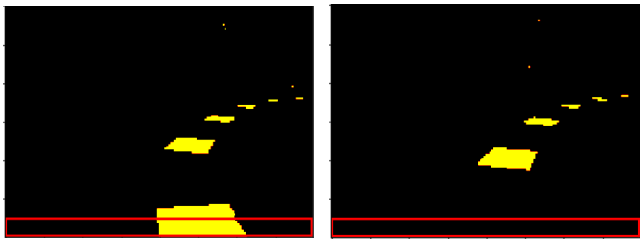
A. Estimate roughly the position in the inertial frame

The position of the vehicle would be estimated roughly based on the number of markers that the vehicle has passed through. Since we have a track whose map is known, presented in Figure 6, the localization of the robot position is reduced from a localization and mapping problem to just localizing the robot's position within this known map. Our strategy is to count the number of blue markers we have passed and deduce our current position based on this information.

Firstly, we need to identify the blue markers. To identify all the blue markers in each image, we use a double color threshold in all three channels of a `hsv` color image. We found that a good lower and upper threshold to detect the blue markers were respectively $lower = [100, 100, 100]$ and $upper = [110, 255, 255]$. We also found that applying the threshold for the `hsv` image instead of a `rgb` image was significantly more robust.

Secondly, we try to implement automatic counting of the number of blue markers we have passed. The counting is implemented by calculating the sum within the red box (Figure 3). The size of the red box is chosen to be the entire image width (such that we will see markers even though we are turning) and 10 pixel height to ensure a certain robustness without looking at to much of the image. If the sum is strictly positive, it means that we are on a blue marker otherwise, if the sum is exactly zero, it means that we are in between two blue markers. If the sum changes from strictly positive to zero, it means that we just left a blue marker and the count goes up by 1.

By counting how many blue markers we have passed, and knowing the position of all the blue markers in the inertial frame, we can make a rough estimation of the position of the robot in the inertia frame. Note that this estimation will only tell us where the robot is up to one blue marker (15 cm).



(a) `In between = False` (b) `In between = True`

Fig. 3: The red rectangle shows what area, we are looking at to check if the vehicle is on a blue marker or between two blue markers.

B. Precisely estimating the position in the inertial frame

In order to estimate the position of vehicle more precisely, we need to calculate our position relative to two blue markers.

After finding the blue markers in the threshold image, we have some very strong gradients in a binary image. We use these to find the contours that forms bounding boxes around each of the blue markers as shown in Figure 4a. The width of each of the bounding boxes can be measured. Moreover, as we know the real-world width of the blue markers, we can calculate the camera's distance to each of the markers using the simplified pinhole model [6],

$$d_i = f\frac{w_i}{W}, \tag{12}$$

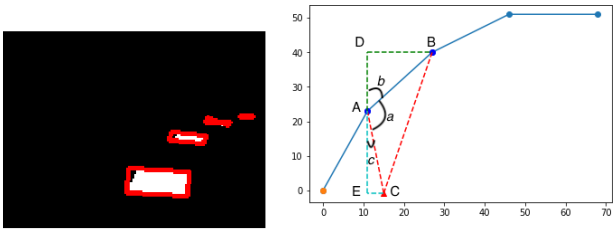
where f is the focal length and W is the true width of each of the blue markers (in our case 7 cm). d_i and w_i represent the distance to the i-th bounding box and the measured pixel width of the i-th bounding box. This gives us an estimation of the distance to the next to blue markers, indicated with the red lines in Figure 4b. As we know the position of A and B, we can calculate the angle a based on a simple geometric relation, presented in (13).

$$a = \cos^{-1}\left(\frac{l_{ab}^2 + l_{ac}^2 - l_{cb}^2}{2 \cdot l_{ab} \cdot l_{ac}}\right) \tag{13}$$

where l_{ab}, l_{ac}, and l_{cb} are the euclidean distances from respectively A to B, A to C, and C to B. Furthermore, we can calculate the angle b using the sinus relations as shown in (14).

$$b = \sin^{-1}(l_{db}/l_{ab}), \tag{14}$$

where l_{db} is the distance between D and B. This allows to calculate the angle $c = \pi - a - b$, and this angle can used to identify the position of the vehicle C in relations to the point A: $[l_{ac} \cdot \sin(c), l_{ac} \cdot \cos(c)]$.



(a) Bounding boxes

(b) Geometric relation between the vehicle and neighboring markers

Fig. 4: The left plot shows bounding boxes whose width is used to calculate the distance to each of them using the pinhole model. The right plot shows the geometry used to calculate the vehicle's relative position to the point A. In this figure, C represents the vehicle's position, A and B indicate the two neighboring blue markers. The red lines are the estimated distance from the vehicle to each of these.

Since, we apriori know the absolute position of the point A and we have calculated the relative position of C with

respect to A, we can merely translate C by A to find the vehicle's position within the map in the inertial frame. This allows us to obtain more accurate position estimation.

V. Experimental Setup

In this section, we will describe our experimental setup.

A. Hardware

The AWS DeepRacer presented in Figure 5 is a 1/18 scale car with a IMU and monocular camera sensor. All four wheels are powered equal and it has front wheel turn. The vehicle has an onboard Ubuntu 16.04 powered computer that can be accessed via `ssh`. The computer is powered by one battery and the vehicle itself is powered by another. The vehicle takes `throttle` and `angle` as input, which represent the velocity and the steering angle in our model respectively.

(a) The vehicle and computer. (b) Only the car.

Fig. 5: Shows the two images of the vehicle under the hood.

B. Calibration

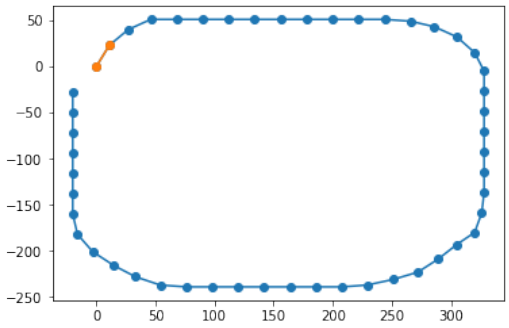
1) Camera: The focal length is calculated through the MATLAB camera calibration app. A series of images of a checkerboard with known square sizes is recorded and a MATLAB calibration app is applied to estimate the most likely homography, yielding all the in the intrinsic camera parameters. The estimated focal length is 127.5062 ± 0.9690 pixels.

2) Motors: To calibrate the AWS DeepRacer vehicle, the vehicle's electronic control system (ECS) and its servomechanism (servo) are calibrated, respectively. Both the servo and ECS accept PWM signals as control input from the vehicle's compute module. The compute module adjusts both of the vehicle's speed and steering angle by changing the duty cycles of the PWM signals. This calibration is operated through an interface from the main navigation pane.

C. RACE TRACK

In order to test the vehicle, we constructed our own race track. The track was approximately 1 meter wide and had a radius of 3 meters. The track consists of four colors: brown (outside the track), green (the border of the track), white (the interior of the track), blue (the center of the track). The blue markers were constructed as 7 cm x 7 cm squares with 15 cm in between. The consistent size and shape was chosen to ease the state estimation. We measured each blue marker's position in the inertia frame. These positions are measured up to 0.5 cm precision, so as expected we observe a small translation error as shown in Figure 6b where loop closure is not achieved.

(a) The constructed track.



(b) The measured map where the orange markers indicates the starting direction.

Fig. 6: The the left we show our race track. Each of the blue markers in the left track correspond to a marker in the 2D plot of the track. The orange markers indicates the starting position.

D. Simulator

Due to the limited time, we haven't implemented the MPC controller on-board the DeepRacer car. Instead, we simulate our MPC controller to track a predefined spline trajectory. Other open source API, such calculating the positions of nearest nodes, are also available [6] [7]. The limitations and difficulties of implementing the MPC controller on-board will be discussed in Section VII.

VI. Results

A. Proportional Controller

With some tuning the proportional controller was successful at keeping the AWS DeepRacer within the track. In a controlled environment, the vehicle reliably made 5 laps of the track without failure.

Primarily, it was important to address the imbalance in the hardware. First, we notice that the steering is skewed to the left, so we adjust the equation for $e_\alpha(t)$ from (6)

$$e_\alpha(t) = \begin{cases} -0.8 & \alpha(t) < -0.2, \\ 1 & \alpha(t) > 0.2, \\ 0 & o.w, \end{cases} \tag{15}$$

[6] https://github.com/AtsushiSakai/PyAdvancedControl
[7] https://github.com/AtsushiSakai/PythonRobotics

where α remains unchanged from (7).

Additionally, the static friction of the vehicle is much higher than the rolling friction. Thus, it requires a high velocity initially, but should be reduced over time. We now define the velocity as a function of time.

$$v(t) = \begin{cases} 0.7 & t < 0.25, \\ 0.5 & t > 0.25. \end{cases} \tag{16}$$

Moreover, the proportional controller suffers in instances where α is noisy, which occurs when the markers are incorrectly recognized. We also note that once the vehicle has left the track, or if the markers are out of view, it is not possible for it to correct this error.

B. MPC Controller

We will evaluate the MPC controller performance on a simulation that tracks several predefined trajectories. We find that the proposed MPC controller is able to operate in stop and go scenarios to track trajectories. Numerical simulation results are illustrated in Figure 7.

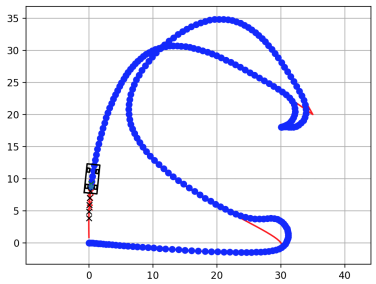


Fig. 7: Simulation results for tracking a spline trajectory. The absolute value of vehicle's velocity is limited between $0.03m/s$ and $0.2m/s$. The horizon number $N = 5$.

We didn't successfully implement the MPC to the DeepRacer car on-board. This require further progress on the motor calibration. Currently, the friction force is very noisy, which makes our controller very unstable to track the given trajectories in a real field.

VII. Discussion and Future Work

This paper, presents the first steps of using the AWS DeepRacer for autonomous driving. There are limitations and difficulties during our implementation. Firstly, we don't have the exact relation between the `throttle` (an equivalent value of PWM signal in the motors) and the vehicle's velocity. This relation should be identified while taking the differences between the coefficient of static and dynamic friction into account. Secondly, our state estimation needs further improvements. Our state estimation is only able to obtain the position of the car, and improvements would be to also estimate is velocity. Furthermore, our approach relies on a pre-defined map of the track. It would desirable to remove this apriori knowledge and estimate a map based on the image information. Some methods of state estimation packages could be applied to fuse the IMU data and the camera's optical flow, for example, VINS [11] and rovio [1].

VIII. Conclusion

In this paper we implemented and tested a proportional controller on the AWS DeepRacer. We found that this simple proportional controller had superior performance to the Deep Reinforcement Learning Controller which is default on the car. We further implemented state estimation based on a pre-defined map of the race track. Finally, we adapted a Model Predictive Controller to the DeepRacer and showed in simulation that the car can be controlled with this controller.

References

[1] M. Bloesch, M. Burri, S. Omari, M. Hutter, and R. Siegwart, "Iterated extended kalman filter based visual-inertial odometry using direct photometric feedback," *The International Journal of Robotics Research*, vol. 36, no. 10, pp. 1053–1072, 2017.

[2] F. Borrelli, A. Bemporad, M. Fodor, and D. Hrovat, "An mpc/hybrid system approach to traction control," *IEEE Transactions on Control Systems Technology*, vol. 14, no. 3, pp. 541–552, 2006.

[3] C. Desjardins and B. Chaib-Draa, "Cooperative adaptive cruise control: A reinforcement learning approach," *IEEE Transactions on intelligent transportation systems*, vol. 12, no. 4, pp. 1248–1260, 2011.

[4] P. Falcone, F. Borrelli, J. Asgari, H. E. Tseng, and D. Hrovat, "Predictive active steering control for autonomous vehicle systems," *IEEE Transactions on control systems technology*, vol. 15, no. 3, pp. 566–580, 2007.

[5] L. Fang and X. Wang, "Lane boundary detection algorithm based on vector fuzzy connectedness," *Cognitive Computation*, vol. 9, no. 5, pp. 634–645, 2017.

[6] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[7] T. Hessburg and M. Tomizuka, "Fuzzy logic control for lateral vehicle guidance," *IEEE control systems magazine*, vol. 14, no. 4, pp. 55–63, 1994.

[8] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, "Kinematic and dynamic vehicle models for autonomous driving control design," in *2015 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2015, pp. 1094–1099.

[9] S. Li, K. Li, R. Rajamani, and J. Wang, "Model predictive multi-objective vehicular adaptive cruise control," *IEEE Transactions on Control Systems Technology*, vol. 19, no. 3, pp. 556–566, 2011.

[10] H. Peng and M. Tomizuka, "Preview control for vehicle lateral guidance in highway automation," *Journal of Dynamic Systems, Measurement, and Control*, vol. 115, no. 4, pp. 679–686, 1993.

[11] T. Qin, P. Li, and S. Shen, "Vins-mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018.

[12] U. Rosolia and F. Borrelli, "Learning model predictive control for iterative tasks. a data-driven control framework," *IEEE Transactions on Automatic Control*, vol. 63, no. 7, pp. 1883–1896, 2018.

[13] U. Rosolia, A. Carvalho, and F. Borrelli, "Autonomous racing using learning model predictive control," in *2017 American Control Conference (ACC)*. IEEE, 2017, pp. 5115–5120.

[14] S. E. Shladover, C. A. Desoer, J. K. Hedrick, M. Tomizuka, J. Walrand, W.-B. Zhang, D. H. McMahon, H. Peng, S. Sheikholeslam, and N. McKeown, "Automated vehicle control developments in the path program," *IEEE Transactions on vehicular technology*, vol. 40, no. 1, pp. 114–130, 1991.

[15] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann *et al.*, "Stanley: The robot that won the darpa grand challenge," *Journal of field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

[16] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.